

Guía de capacitación Git/Nexus

Preparación del Ambiente

1. Instalar Git, archivo `Git-2.18.0-64-bit.exe`.
2. Abrir un `git shell` y configurar usuario y mail:

```
# usuario y correo son necesarios
git config --global user.name "Nombre Usuario"
git config --global user.email "mail@usuario.com"

# mergetool es opcional pero deseable para mostrar por cli cómo hacer un
merge
git config --global merge.tool tortoisemerge
```

3. Abrir un `git shell` y generar un par de claves ssh:

```
ssh-keygen
```

4. Subir la clave ssh generada a gitlab:

```
# copiar la salida del siguiente comando y
# pegarla en https://HOST_GITLAB/profile/keys
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDdbWnXdid+.....
```


5. Instalar TortoiseGit, archivo `TortoiseGit-2.7.0.0-64bit.msi`
6. Generar clave gpg (desde un shell). **Opcional:**

```
gpg --gen-key
```

Primeros pasos

El objetivo de estos primeros pasos es ver los comandos `clone`, `add`, `commit`, `push`, `status`, `fetch`, `merge` y `log`. Tanto desde línea de comandos como utilizando `TortoiseGit`. Se adjuntan una serie de archivos para hacer modificaciones y evitar tener que crear y cambiar archivos a mano. En el directorio `versiones` vas a encontrar una serie de carpetas numeradas que serán referenciadas en los pasos que sea necesario agregar o modificar archivos.

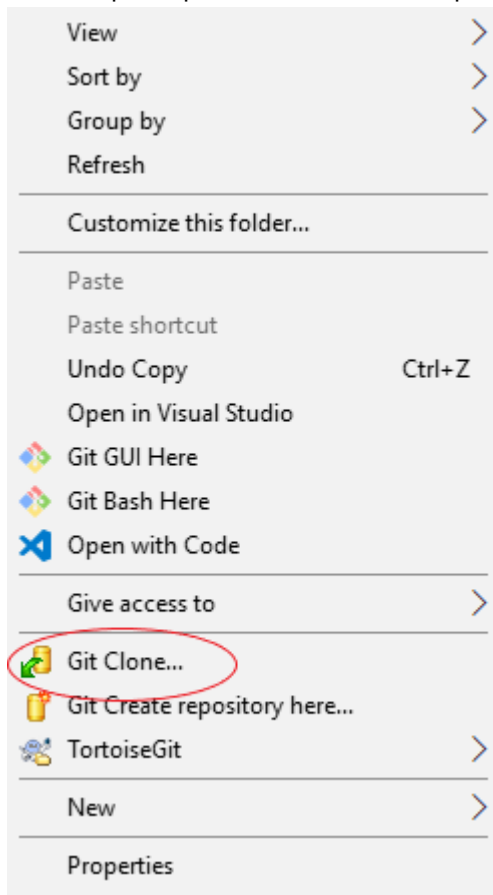
1. Crear proyecto nuevo en GitLab. Puede crearse privado al usuario o en un grupo en el cual el usuario

A green rectangular button with the text "New project" in white, followed by a small downward-pointing arrow.

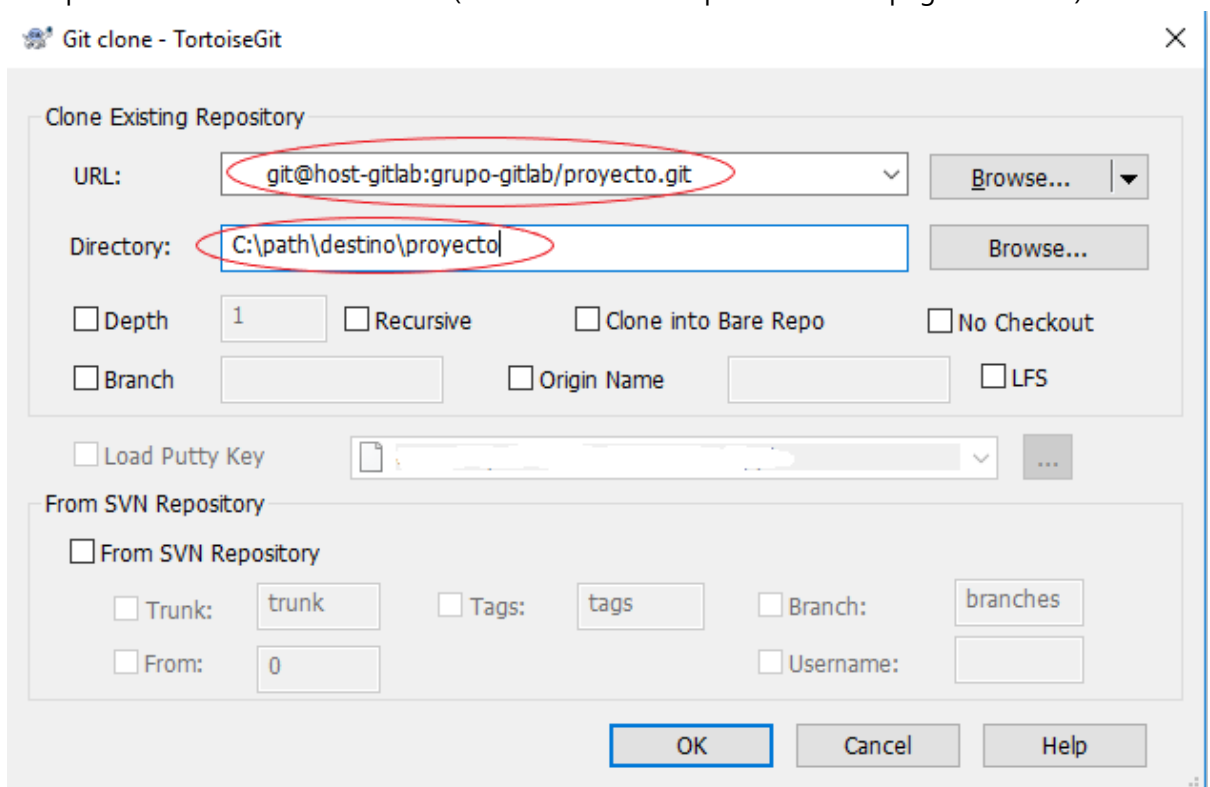
tenga acceso. Ejemplo, en <https://host-gitlab/nombre-grupo>:

2. Hacer un **clone** del proyecto de GitLab creado usando TortoiseGit.

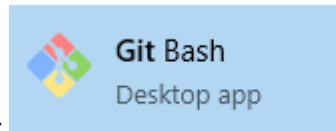
1. En la carpeta que se desea clonar el proyecto presionar botón derecho y elegir **Git Clone...**



2. Completar con los datos necesarios (obtener la url del repositorio de la página [GitLab](#))



3. Hacer el mismo **clone** pero usando línea de comandos.

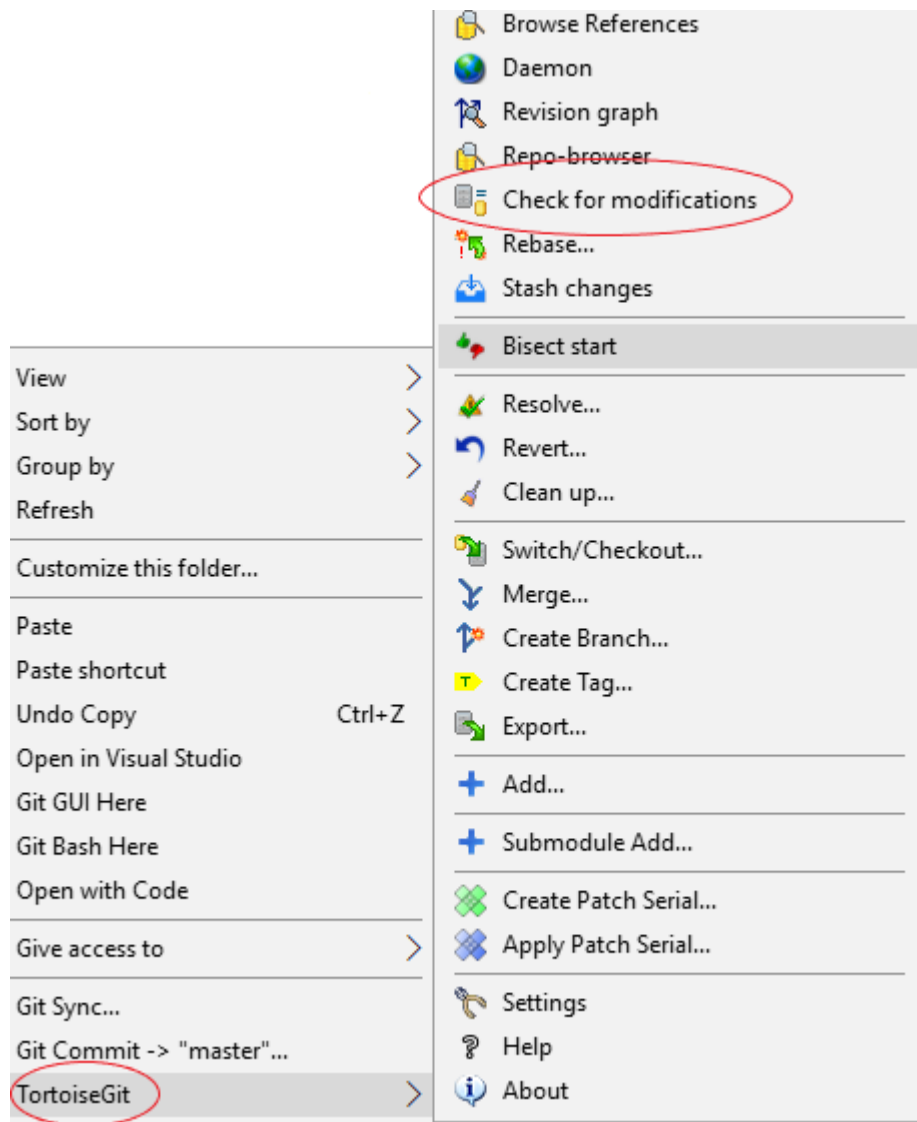


- Abrir un **Git Bash**:
- Ejecutar el comando:

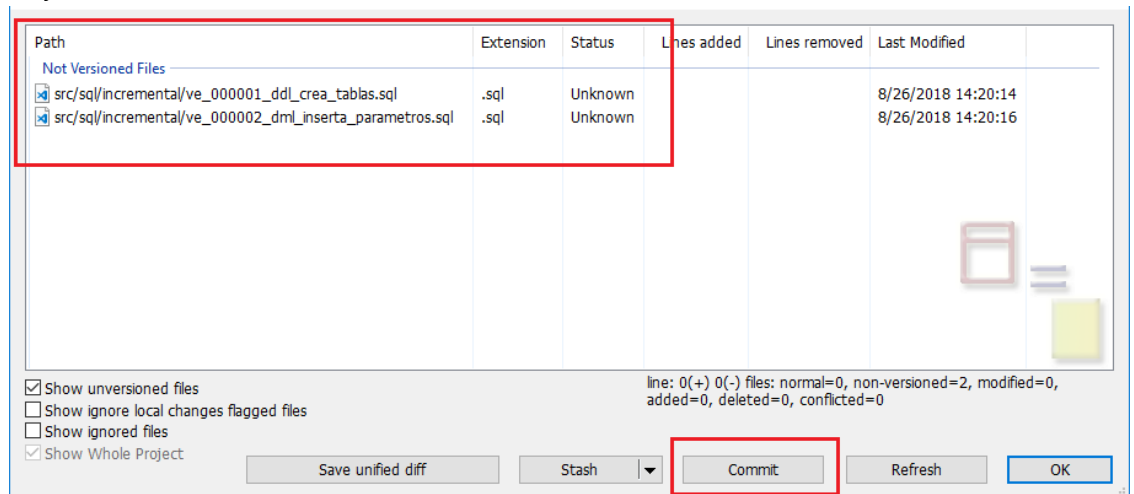
```
$ git clone git@host-gitlab.com:grupo-gitlab/proyecto.git carpeta-destino
Cloning into 'carpeta-destino'...
Enter passphrase for key:
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 38 (delta 10), reused 0 (delta 0)
Receiving objects: 100% (38/38), done.
Resolving deltas: 100% (10/10), done.
```

4. Agregar contenido al proyecto clonado (sacar los archivos de **versiones/001**).

- Hacer un **add** desde **TortoiseGit**. Observar el área de **staging** antes y después del **add**.
 - Primero visualizar los cambios:

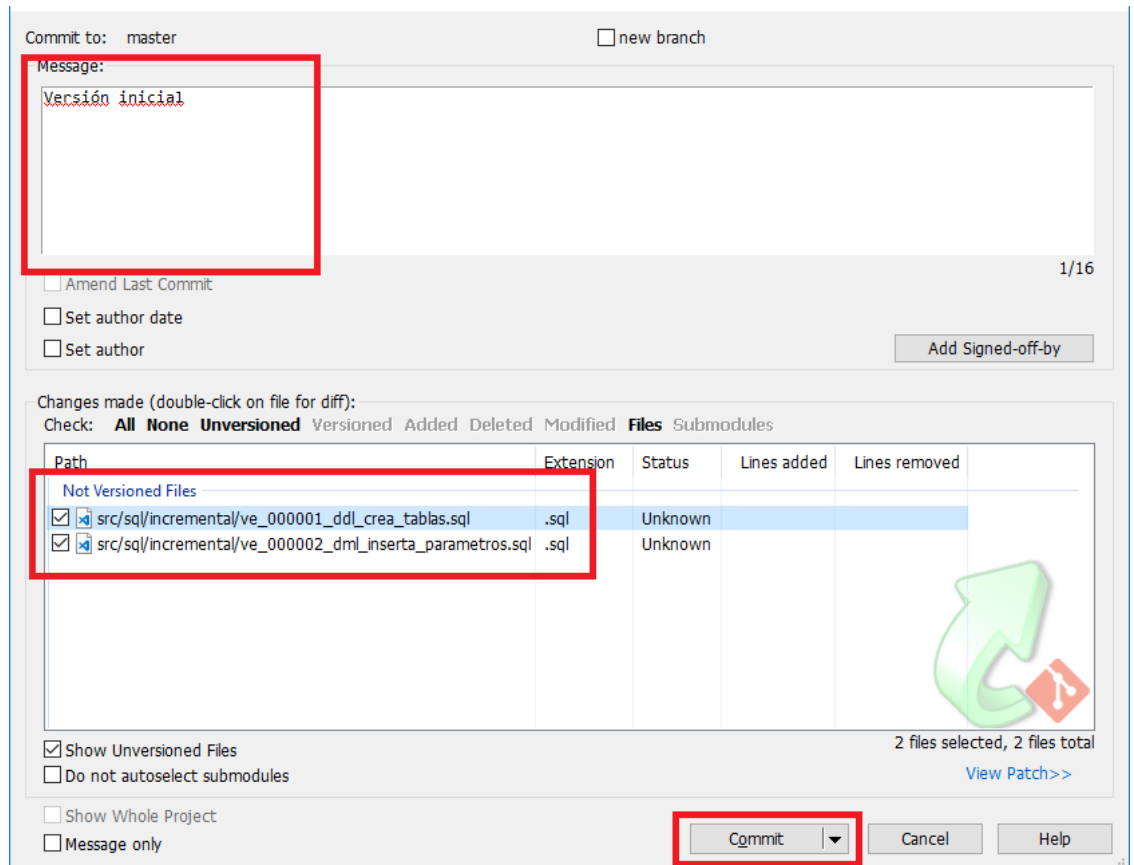


- Hay dos archivos nuevos (estado *Unknown*):



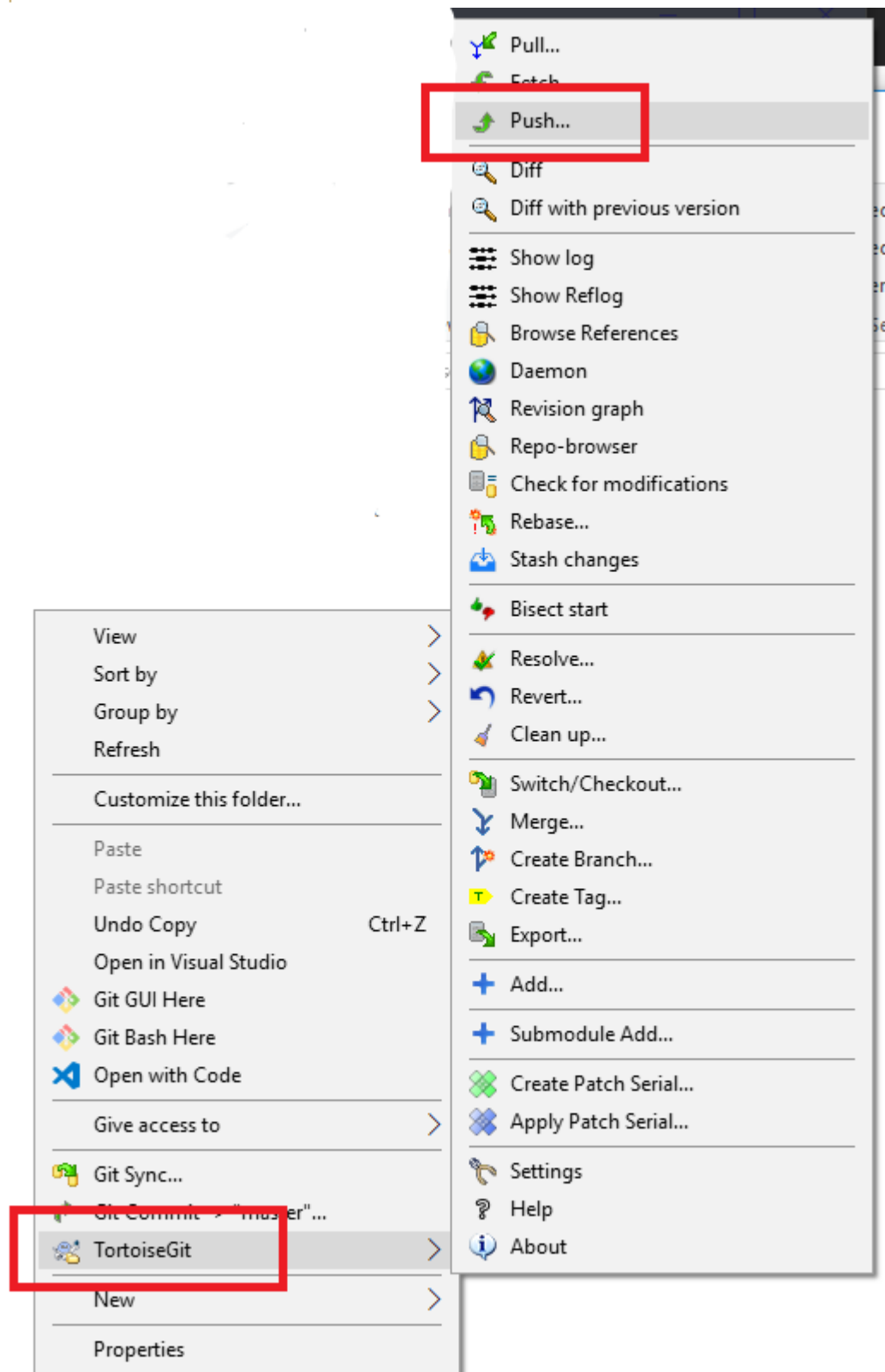
- Hacer **commit** desde TortoiseGit.

- Seleccionar los archivos a comitear. Tener en cuenta buenas prácticas para realizar comentarios.



- Ir a GitLab: aún no se ven los cambios comiteados.

- Hacer **push** desde TortoiseGit. En GitLab si se ven cambios.



5. Hacer un **status** en el repositorio que se clonó usando **Git Bash**. Analizar la salida.

```
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

6. Hacer **fetch** desde el repositorio que se clonó usando **Git Bash**. La salida de git indica que se recibieron cambios del repositorio remoto, pero aún no se ven impactados los cambios en forma local:

```
$ git fetch
Enter passphrase for key:
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 7 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (7/7), done.
From gitlab.com:ventanilla/ve-lector-sql
* [new branch]      master    -> origin/master
```

7. Hacer un **merge** desde línea de comandos. Analizar la salida. Ya se ven los cambios y git indica que estamos sincronizados con el repositorio remoto, en **GitLab**, nombrado como **origin/master**:

```
$ ls

$ git merge origin/master

$ ls
src/

$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

8. Agregar más contenido al proyecto (sacar los archivos de **versiones/002**).

- Visualizar estado desde línea de comandos. Se ven cambios *Untracked*, es decir, que aún no se han incorporado al repositorio local:

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
(use "git add <file>..." to include in what will be committed)

    src/sql/incremental/ve_000003_dml_inserta_datos_iniciales.sql
...
```

- Hacer un **add** desde línea de comandos. Los archivos están en el área de **staging**, listos para ser *comiteados*:

```
$ git add .
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

       new file:   src/sql/incremental/ve_000003_dml_inserta_datos_iniciales.sql
```

- Hacer un **commit** desde línea de comandos. No se ve en GitLab. El **commit** implica que los cambios quedan impactados en el repositorio local, pero aún no los hemos subido a **GitLab**:

```
$ git commit -m "Agrega inserts con datos iniciales"
[master 959dc6a] Agrega inserts con datos iniciales
1 file changed, 3 insertions(+)
create mode 100644
src/sql/incremental/ve_000003_dml_inserta_datos_iniciales.sql
```

- Hacer un **push** desde línea de comandos. Ahora sí, ya se ve en GitLab:

```
$ git push origin master
Enter passphrase for key:
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 541 bytes | 180.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0)
To gitlab.com:ventanilla/ve-lector-sql.git
ad6d0f5..959dc6a  master -> master
```

- Mostrar la historia de cambios con el comando **log**:

```
$ git log
commit 959dc6abd816b34d7cb809f3f44c644cf915c17a (HEAD -> master,
origin/master)
Author: Nombre Usuario <mail@algo.com>
Date:   Sun Aug 26 17:44:48 2018 -0300

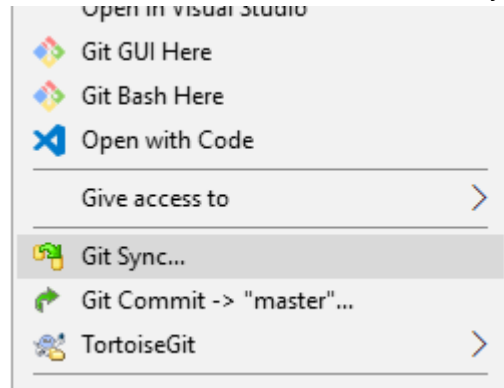
    Agrega inserts con datos iniciales

commit ad6d0f52b03fd8529265f9476ed9fc54085ef430
Author: Nombre Usuario <mail@algo.com>
Date:   Sun Aug 26 17:20:06 2018 -0300
```

Versión inicial

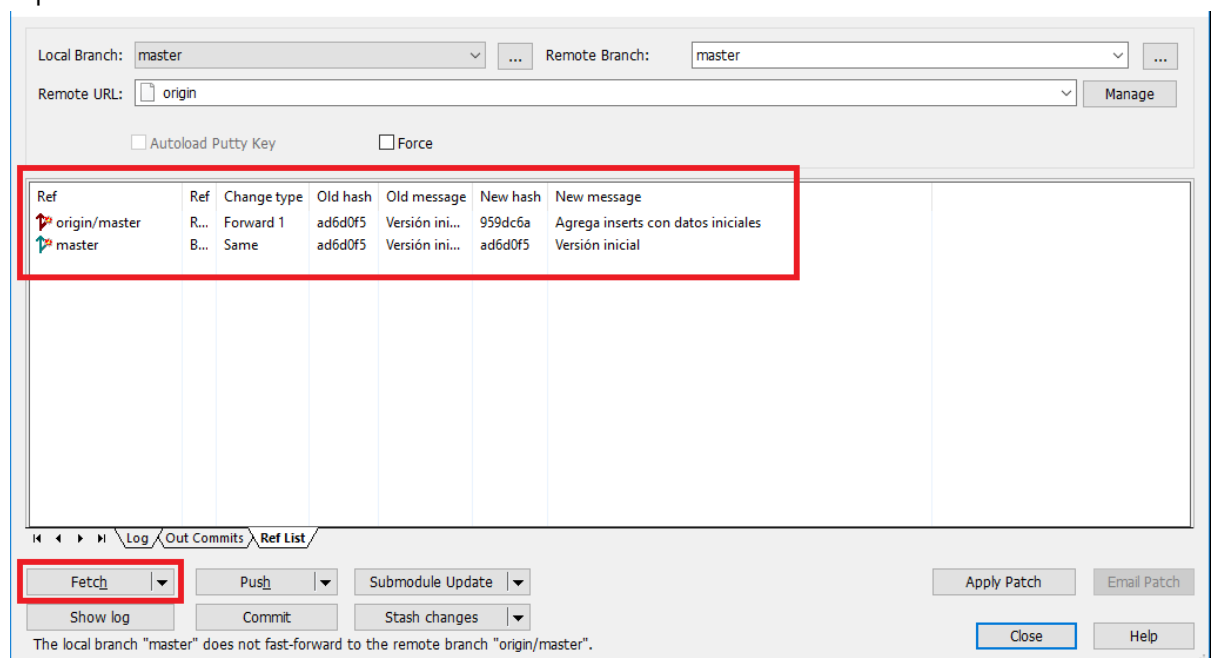
9. Sincronizar el repositorio de TortoiseGit:

- Ingresar a la opción de sincronizar, posicionándose en la carpeta del proyecto que estamos manejando con TortoiseGit, haciendo *click* con el botón derecho del *mouse* y seleccionando



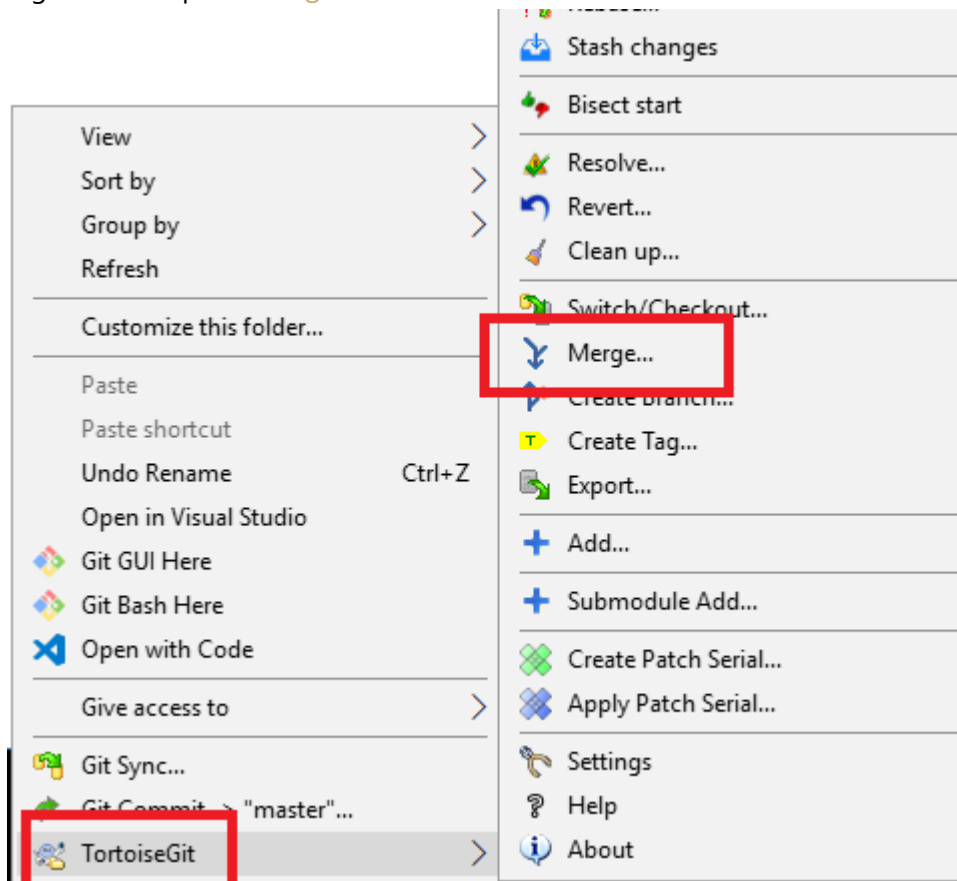
Git Sync... en el menú contextual:

- En la ventana que aparece seleccionar **fetch** (si se selecciona **pull** no se podrán ver los cambios antes de sincronizar). Luego de ejecutado el **fetch** se podrán ver las novedades respecto al repositorio remoto:

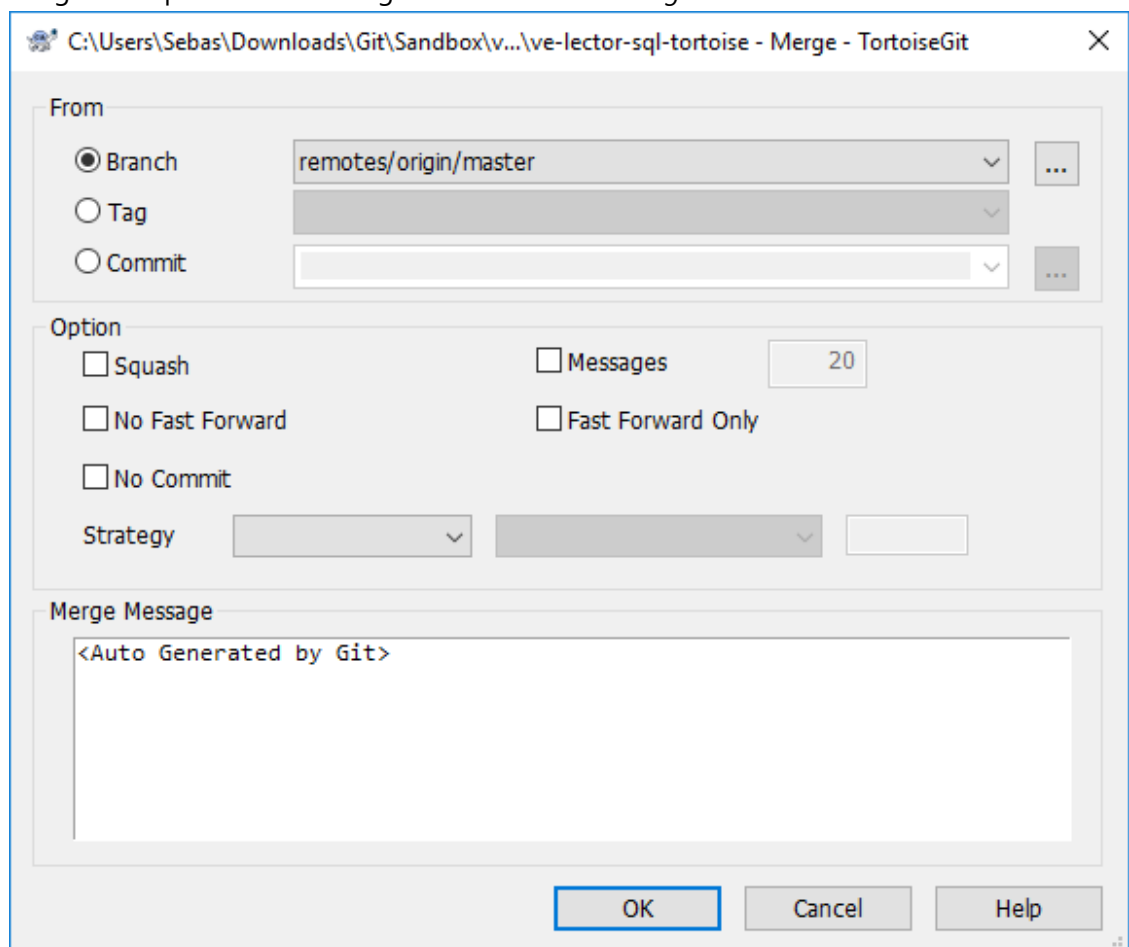


- Realizar el **merge**:

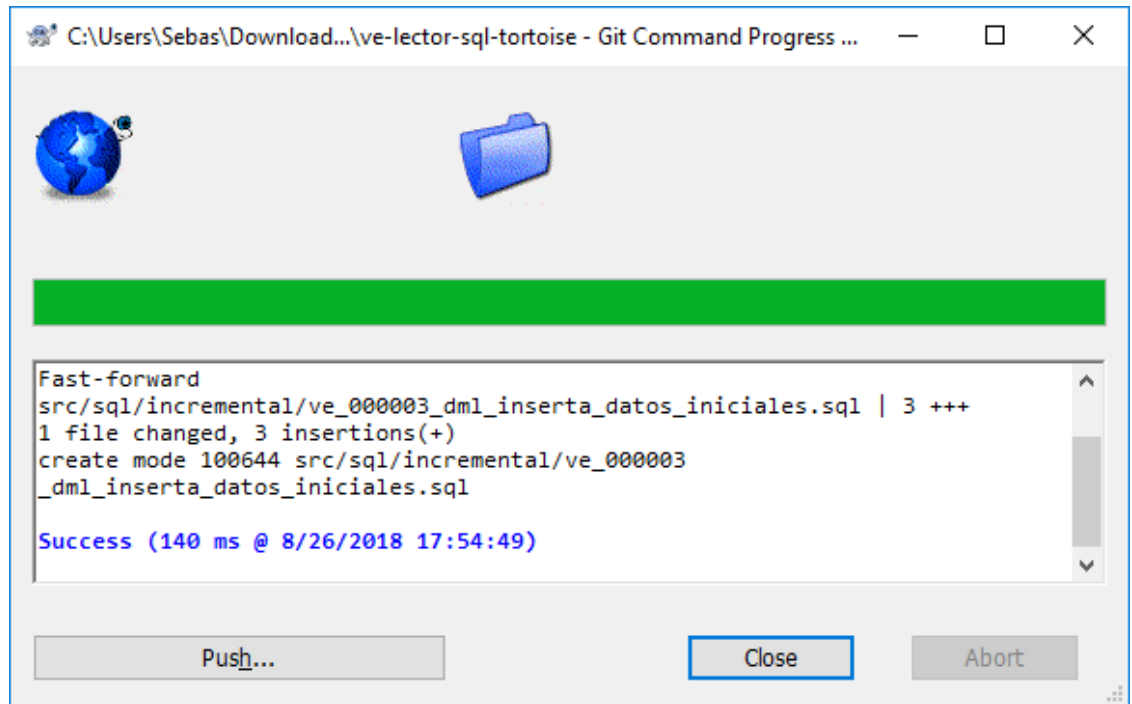
- Ingresar a la opción **merge**:



- Asegurarse que la rama a mergear es el *master* de *origin*:



- Observar el resultado:



Hasta el momento se tienen dos copias del mismo repositorio, una que se maneja usando **TortoiseGit** y otra que se maneja por línea de comandos, usando **Git Bash**. Si se siguieron todos los pasos ambos están sincronizados, y también lo está el repositorio remoto en **GitLab**.

Cualquier duda que se presente con los comandos utilizados, desde una consola **Git Bash** escribir **git help <comando>** para obtener ayuda en línea:

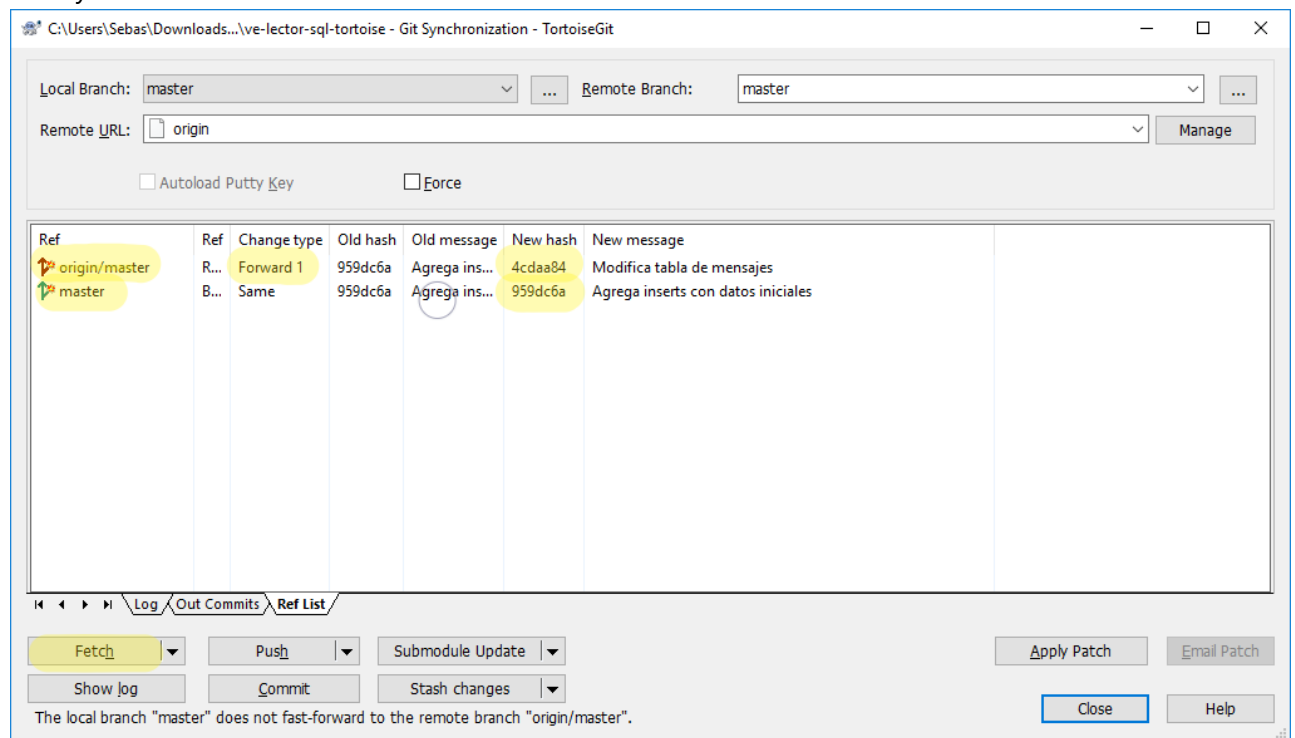
```
$ git help merge
# abrirá un navegador con la información solicitada
```

O bien en su defecto consultar la ayuda de **TortoiseGit** que está orientada a cómo usar los comandos desde el propio **TortoiseGit**.

Actualizar repositorio con cambios hechos por otro usuario

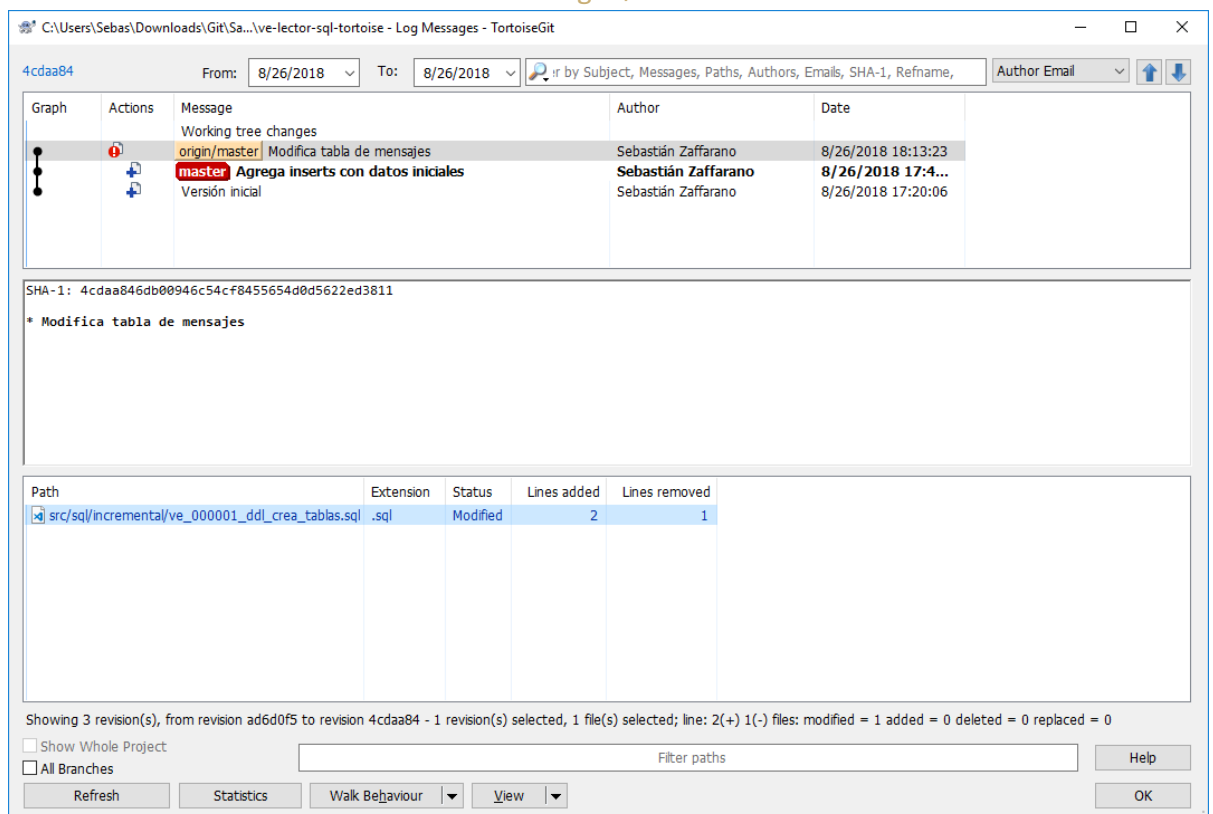
1. Agregar contenido al repositorio que se está manejando por **Git Bash** (sacar los archivos de **versiones/003**). Seguir los mismos pasos que ya se hicieron antes para la operación de **add**.
2. Hacer un **commit** para incorporar los cambios en el repositorio manejado por **Git Bash**.
3. Hacer un **push** para subir los cambios en el repositorio manejado por **Git Bash**. Verificar que los archivos se visualicen en **GitLab**.
4. En el repositorio manejado por **TortoiseGit** realizar un **sync** como se hizo previamente (haciendo **fetch**), y observar la salida. Vemos que el repositorio de **GitLab** (**origin/master**) está un **commit** adelante que nuestro repositorio local (**master**). Se observan también los **hashs** de ambos commits, el

local y el remoto:

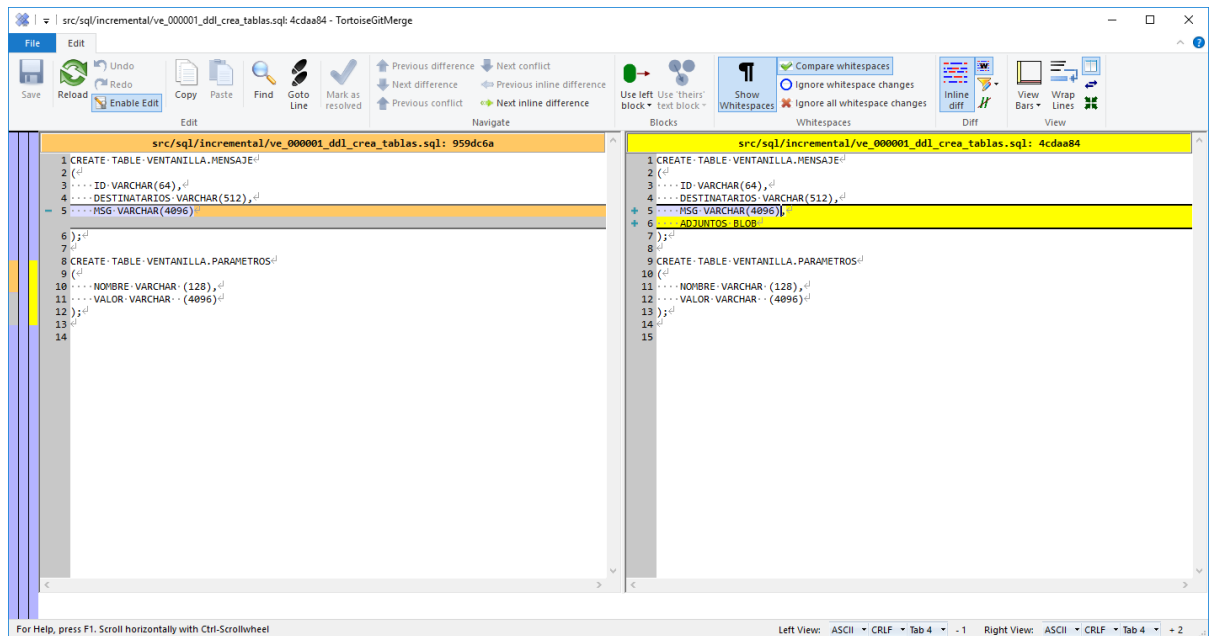


5. Hacer un **merge** observando qué cambio:

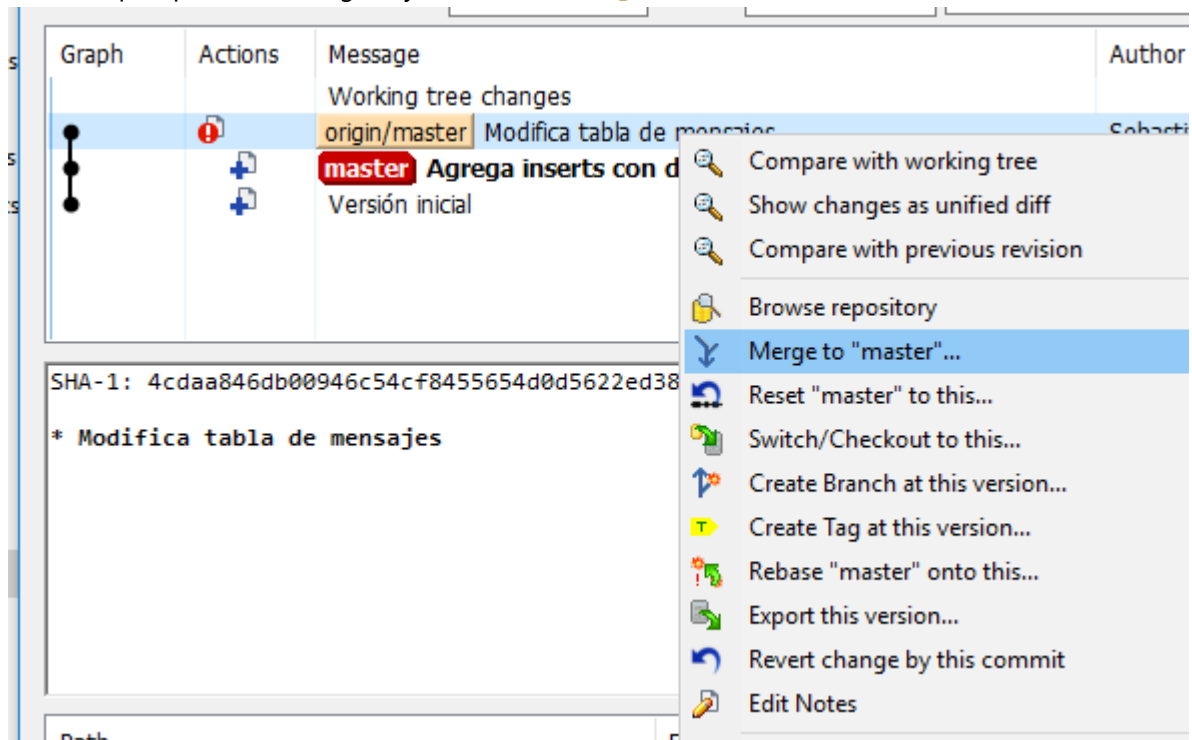
- El **master** local está un commit detrás del **origin/master**



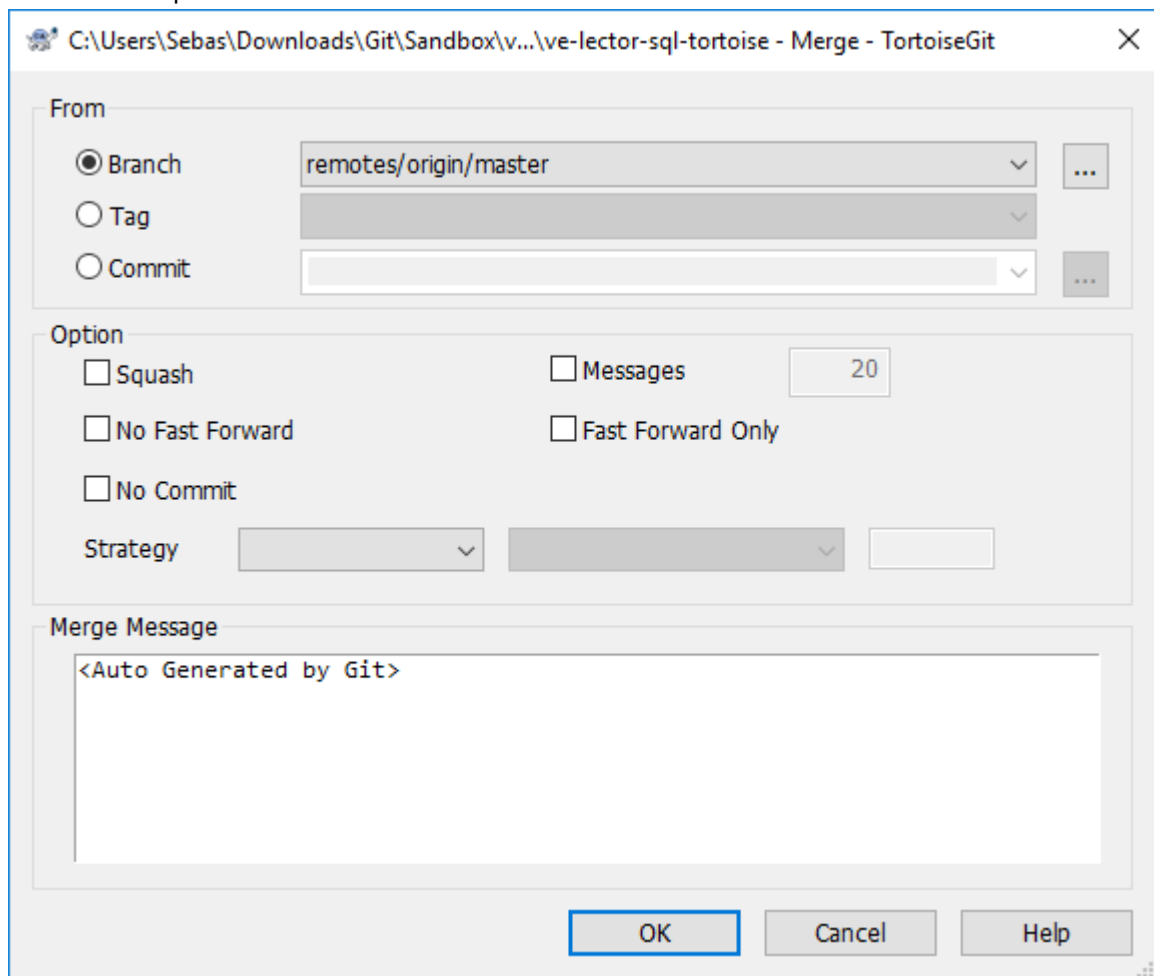
- Si se hace doble click en cualquiera de los archivos listados (en el ejemplo hay sólo uno), se visualizarán los cambios, en este caso, se agregó una columna a la definición de la tabla:



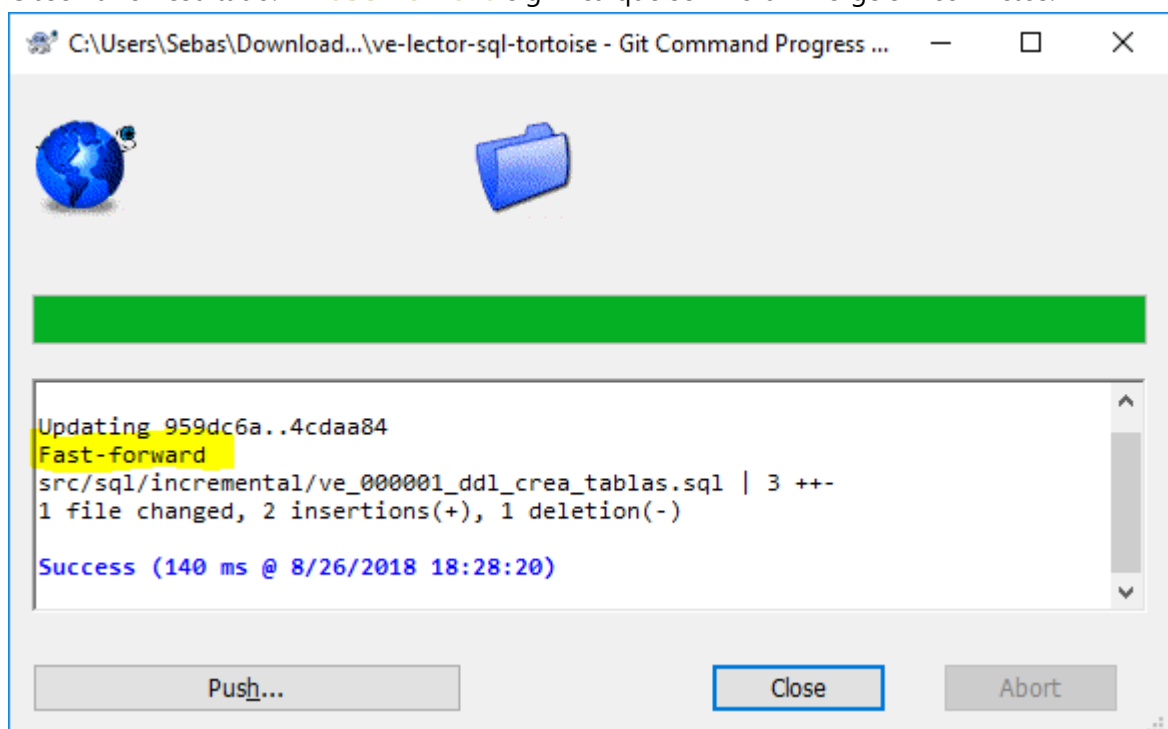
- Si los cambios son correctos, para impactarlos en nuestro repositorio local, posicionarse en el **commit** que queremos *mergear* y seleccionar **Merge to "master"**:



- Presionar aceptar:



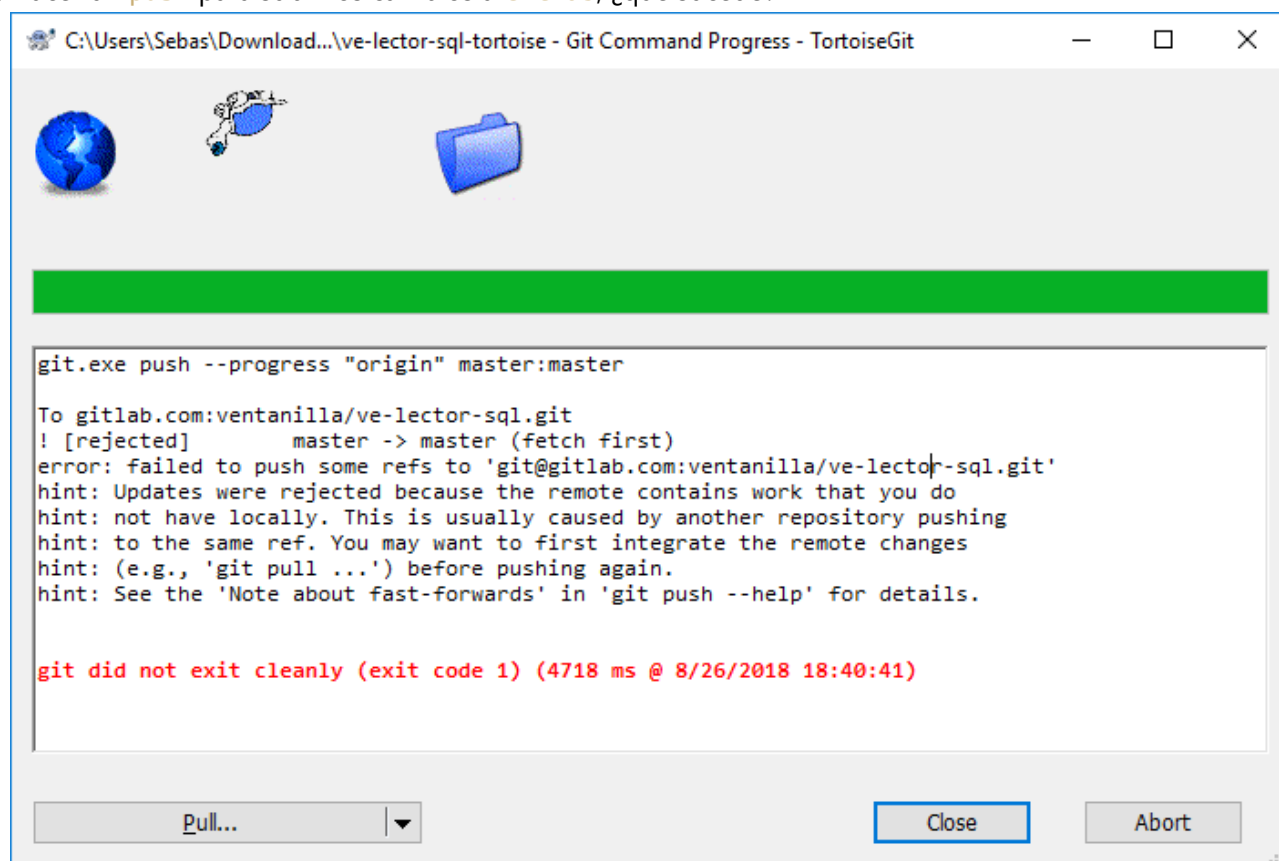
- Observar el resultado. El **Fast-forward** significa que se hizo un merge sin conflictos.



6. Verificar que ambos repositorios locales estén actualizados y sincronizados con **GitLab** (recordar el procedimiento viendo más arriba).

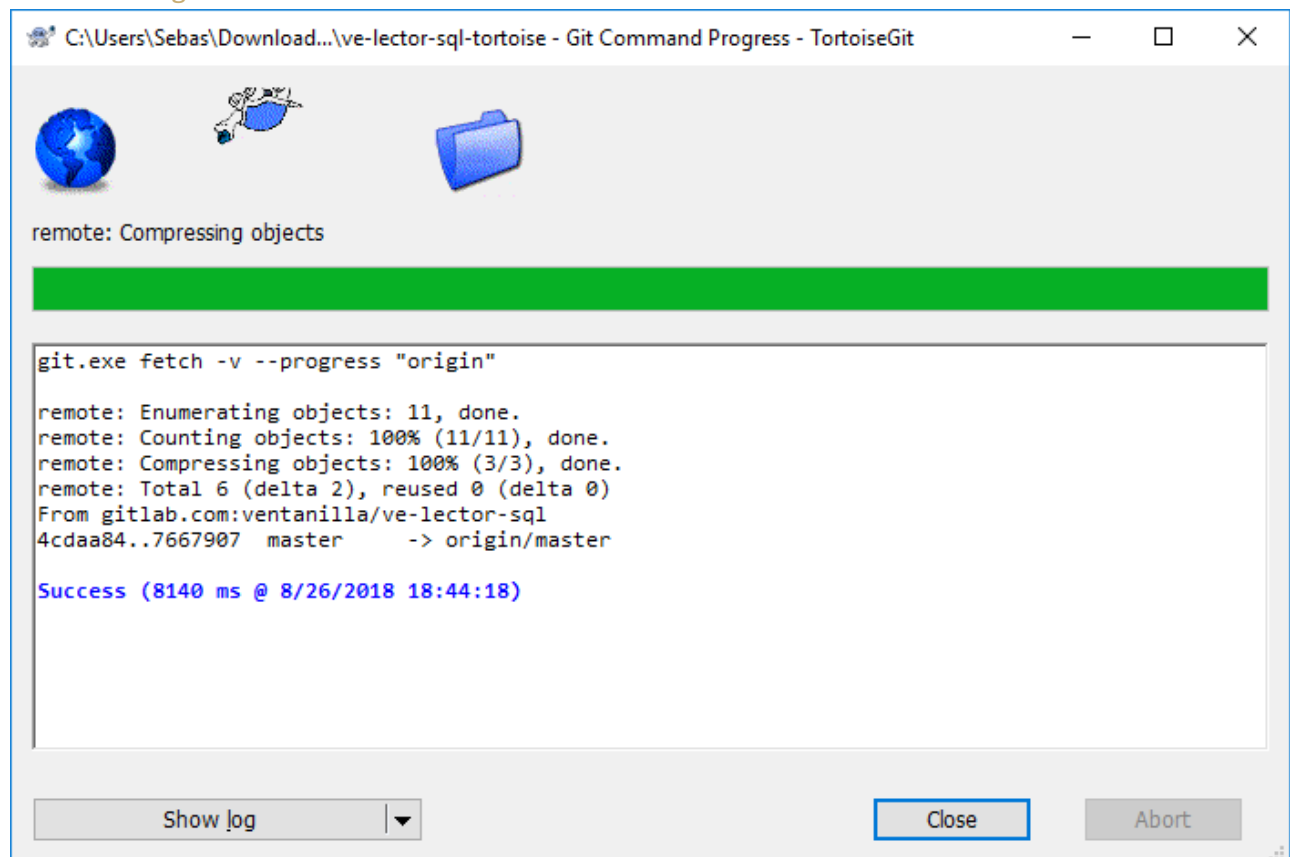
Resolver conflictos

1. Agregar contenido al repositorio que se está manejando por **Git Bash** (sacar los archivos de **versiones/004**). Seguir los mismos pasos que ya se hicieron antes para la operación de **add**.
2. Hacer un **commit** para incorporar los cambios en el repositorio manejado por **Git Bash**.
3. Hacer un **push** para subir los cambios en el repositorio manejado por **Git Bash**. Verificar que los archivos se visualicen en **GitLab**.
4. Agregar contenido al repositorio que se está manejando por **TortoiseGit** (sacar los archivos de **versiones/005**). Como no se sincronizó este repositorio local respecto a **GitLab**, se van a generar conflictos pues hemos modificado un archivo que fue previamente modificado en los pasos previos. Es **muy importante** mantener el repositorio local sincronizado para evitar inconvenientes como los de este paso. Acá no lo hacemos para producir conflictos y ver cómo solucionarlos.
5. Hacer un **commit** para incorporar los cambios en el repositorio manejado por **TortoiseGit**.
6. Hacer un **push** para subir los cambios a **GitLab**, ¿qué sucede?

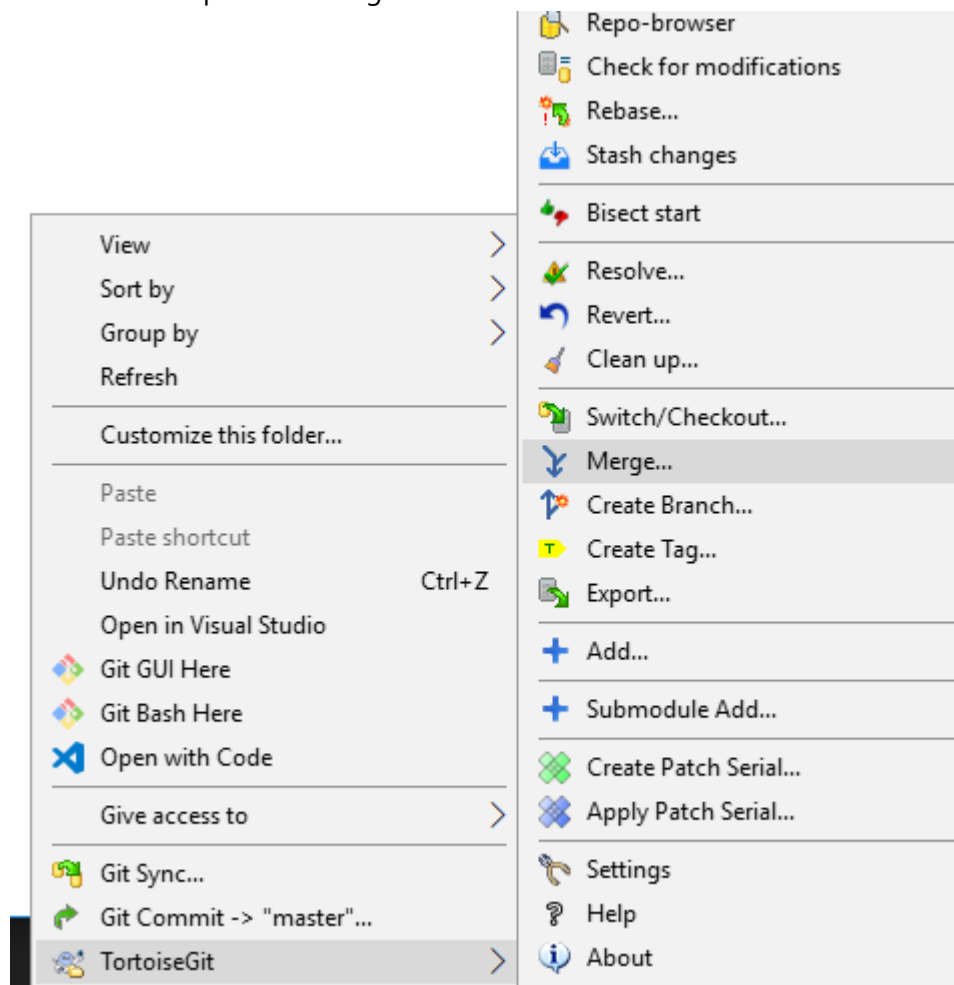


7. Es necesario resolver el conflicto. Sincronizar el repositorio local haciendo un **fetch**, tal como se hizo anteriormente. Se puede ver que se recibieron cambios hechos y subidos en el servidor que requieren

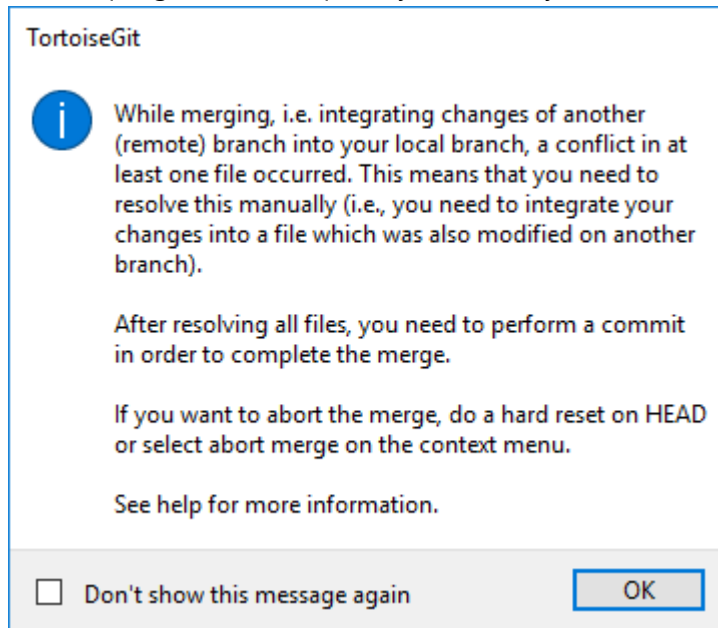
hacer un **merge** localmente:



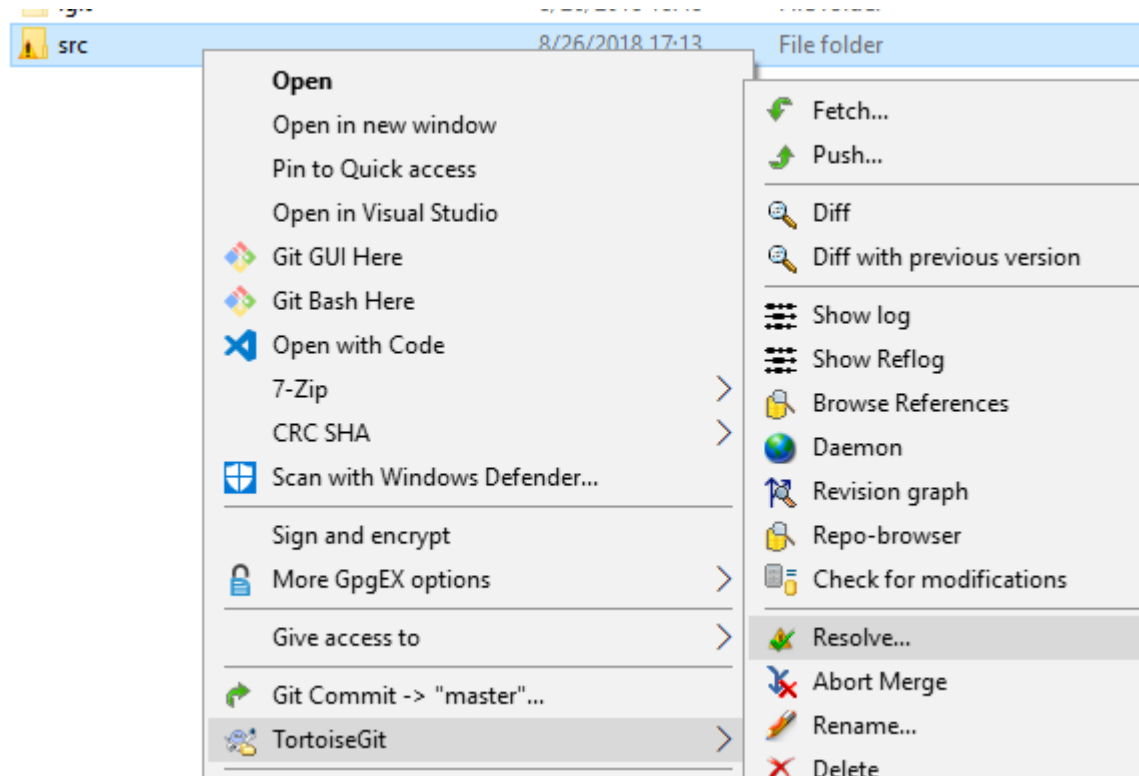
8. Seleccionar la opción de merge:



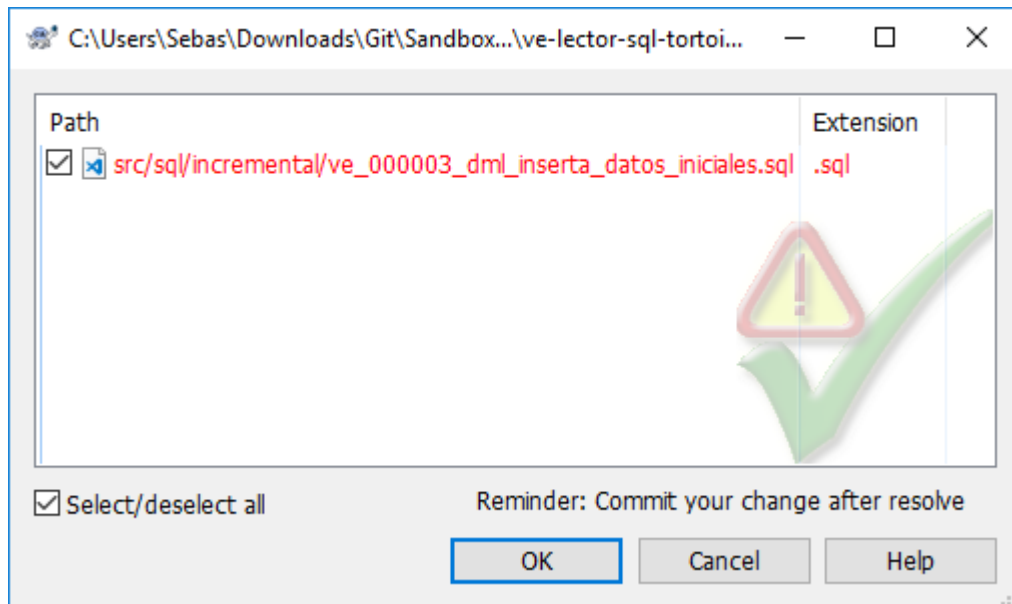
9. Vemos que git nos avisa que hay conflictos y es necesario intervención manual:



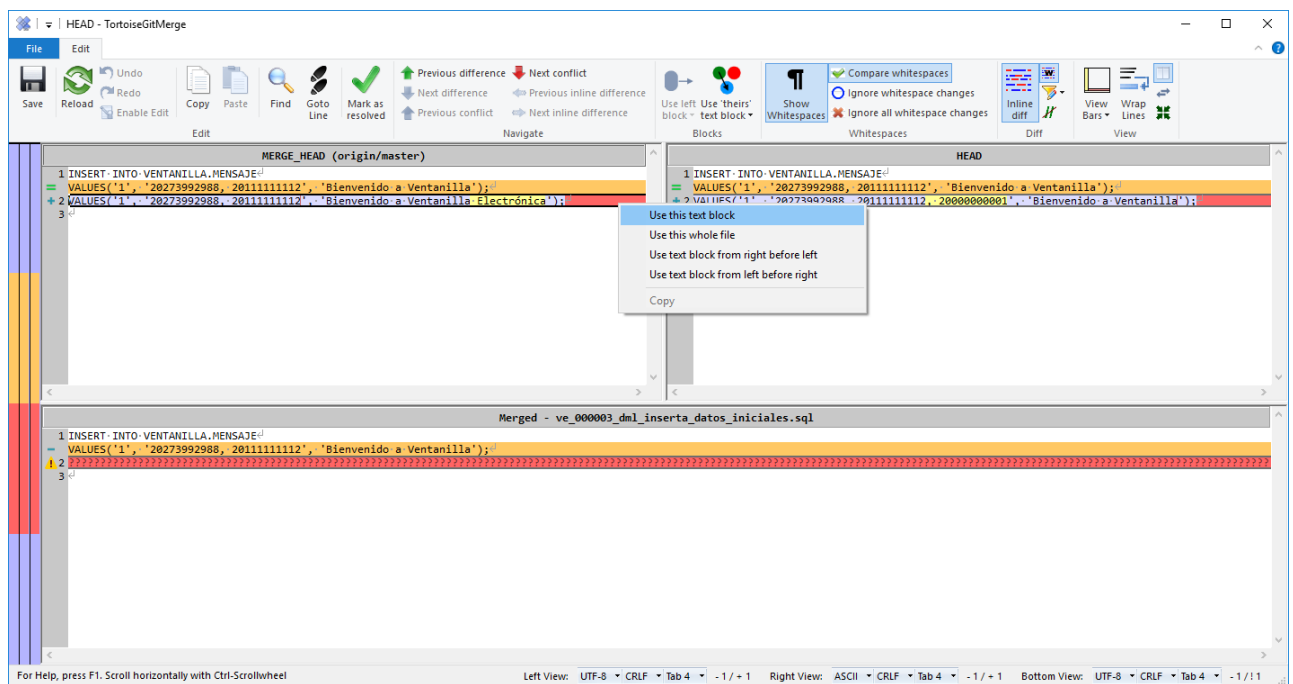
10. Seleccionamos la opción **Resolve**. Observar que el ícono en la carpeta que contiene los archivos con conflicto cambió:



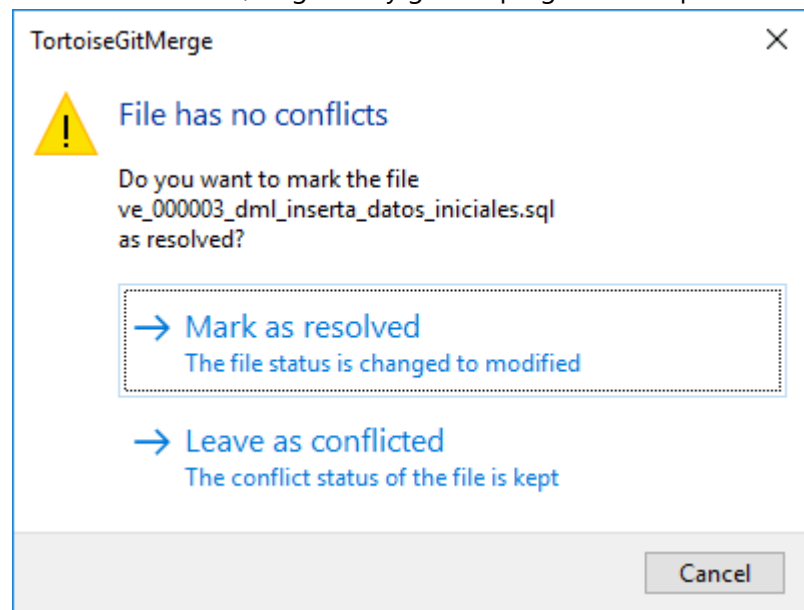
11. Aparecerá una ventana listando todos los archivos con conflictos (en el ejemplo sólo hay uno). Al dar doble click aparecerá una ventana mostrando las diferencias a resolver.



12. La ventana muestra que dos *commits* independientes modificaron la misma línea. Se puede optar por elegir la versión local, la versión remota, o formar a mano una nueva versión con parte de los cambios de cada *commit*.

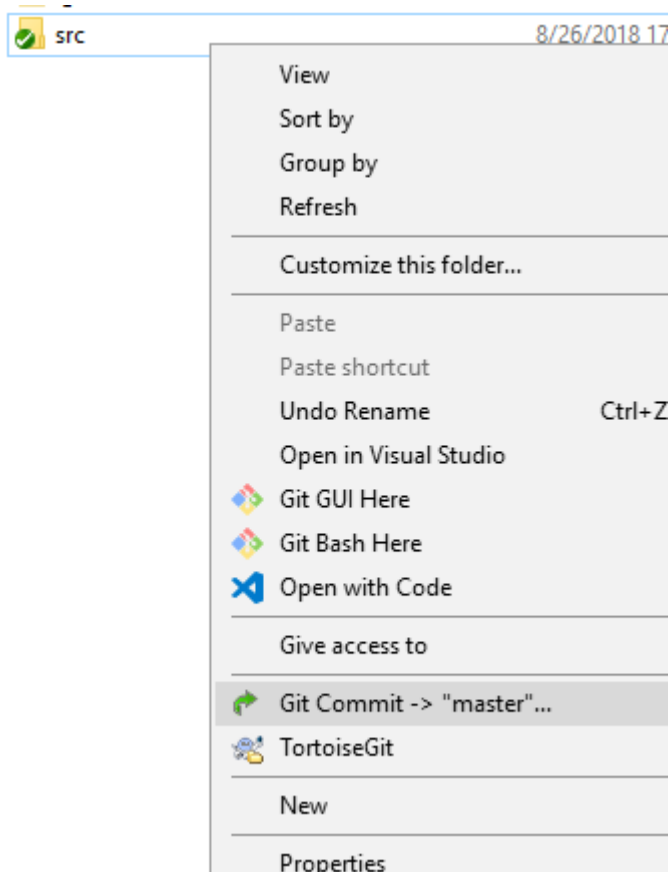


13. Cuando se llega a una versión que no tiene conflictos, se guarda y git nos preguntará si queremos



marcar como resuelto el conflicto.

14. Si hubiese más archivos, repetir los pasos hasta que no quede ninguno. Luego de esto hacer un **commit** para dar por finalizado el **merge**. Observar cómo el ícono de la carpeta cambió indicando que no hay



conflictos.

15. Ahora si, hacer un push para subir a **GitLab** los cambios.

16. Sincronizar el repositorio que manejamos usando **Git Bash** para que quede igual que el resto de los repositorios.

Tags

@TODO

Branches

@ TODO

Posibles problemas

Si no aparecen los íconos de Tortoise en Explorer, es posible que haga falta editar la siguiente entrada de la registry windows:

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ShellIconOverlayIdentifiers`. Para más información ver [este thread](#).